GPU Performance and Power Consumption Analysis: A DCT based denoising application

Martín Pi Puig [1], Laura De Giusti [1], Marcelo Naiouf [1], Armando De Giusti [1, 2]

[1]Instituto de Investigación en Informática LIDI (III-LIDI), Facultad de Informática, Universidad Nacional de La Plata, 50 y 120 2do piso, La Plata, Argentina.
[2]CONICET – Consejo Nacional de Investigaciones Científicas y Técnicas, CABA, Argentina
{mpipuig, ldgiusti, mnaiouf, degiusti}@lidi.info.unlp.edu.ar

Abstract. It is known that energy and power consumption are becoming serious metrics in the design of high performance workstations because of heat dissipation problems. In the last years, GPU accelerators have been integrating many of these expensive systems despite they are embedding more and more transistors on their chips producing a quick increase of power consumption requirements. This paper analyzes an image processing application, in particular a Discrete Cosine Transform denoising algorithm, in terms of CPU and GPU performance and energy consumption. Specifically, we want to compare single-threaded and multithreaded CPU versions with a GPU version, and characterize the execution time, true instant power and average energy consumption to deflate the idea that GPUs are non-green computing devices.

Keywords: Denoising, Power, Energy, GPU, NVML, RAPL.

1. Introduction

The computing scenario has changed substantially since the introduction of accelerators, principally GPGPU (short for general purpose computing on graphics processing units). Therefore, the number of devices with GPUs and the amount of GPU accelerated applications increased more and more over the past years.

These devices have drawn the attention of the research community because they have a great computational power next to a high memory bandwidth, and are formidably suited for massively data parallel computation (Single Instruction Multiple Threads applications). Coming along with these features, the energy consumption of GPU containers like high performance workstations and personal computers became a real problem [1]. Some direct consequences of its higher power consumption are growing dissipation of heat, more complex cooling solutions, and noisier fans [2].

As a result, power dissipation must be reduced without losing computing performance. Further, the peak power of latest Nvidia and AMD GPUs is as high as 300W, while a typical CPU consumes only 80W at TDP. This does not indicate that

the GPU has lower energy efficiency since the increasing advantage in performance can offset the larger power consumption [3][4]. For instance, in the June 2017 Green 500 ranking, 8 of 10 top computer systems incorporate Nvidia accelerators.

Historically, GPGPU researching has focused primary on accelerating scientific applications [5] such as physical simulations, medical analysis, image and video processing. This work not only accelerates an image processing algorithm via GPU, it also makes a description of performance, power and energy consumption of CPU and GPU.

Generally, the energy quantification process relies on two approaches: hardware based and software based measurement. In first, a physical measuring device is attached between the power supply and the many-core device. Then, the electrical power can be computed by multiplying current and voltage. Also, energy consumption can be determined by integrating the power over total execution time [6]. On the other hand, latest high-end Nvidia GPUs provide the possibility to read the current power consumption by software through the Nvidia Management Library (NVML) [7]. In addition, Intel CPUs present a set of counters providing energy and power consumption information through Running Average Power Limit (RAPL) software.

In this case, we raise the image denoising issue, where zero-mean white Gaussian additive noise must be removed from a given set of images. The approach taken is based on a simple Discrete Cosine Transform (DCT) hard thresholding with a given scale factor over noise standard deviation.

Specifically, we implement a serial CPU version of the algorithm and two additional versions: a multithreaded CPU version and a GPU version, where the computation runs exclusively on the accelerator. Also, we evaluate their performance, instant power, and energy consumption.

The rest of the paper is organized as follows. Section 2 presents the background for the proposed work. Section 3 describes the main performance and power results and finally Section 4 makes a brief conclusion of obtained data.

2. Background

This section presents some background material in order to put our research in perspective, particularly the DCT denoising theory, a brief review of power consumption measuring scenarios and a description of the chosen experiments.

2.1 DCT Image denoising

This work addresses the classic denoising problem: an image is contamined by noise in its acquisition or transmission. Generally, pictures taken with both digital cameras and conventional film cameras will pick up noise from a variety of sources. Further use of these images will often require that the noise be removed for aesthetic purposes as in artistic work or marketing, or for practical purposes such as computer vision.

Signal denoising aims to estimate the original image while retaining as much as possible the important figure features. Thus,

$$y(i,j) = x(i,j) + n(i,j), \tag{1}$$

where $y(i,j)$ is the examined value, $x(i,j)$ is the original value and $n(i,j)$ is the noise perturbation at pixel i,j. Besides, noise is often modeled as a zero-mean Gaussian process with a given standard deviation σ. This signal is independent of the original image and it commonly appears as high frequency coefficients, but useful signals appear as either low frequency or smoother details.

When we decompose a given signal using DCT, we are left with a set of spectral coefficients that correlates to details in the image. If these details are small enough, they can be omitted without substantially affecting the main picture quality. Additionally, these values are often those associated with noise. Therefore, by setting these coefficients to zero, we are essentially reducing/killing the noise. This becomes the basic concept behind thresholding: setting all frequency coefficients that are less than a specific threshold to zero and use these coefficients in an inverse DCT process to reconstruct the image [8].

So, the denoising process could be reduced to: apply DCT to (1) to get the spectral coefficients $Y(i,j)$. Eventually tweak these coefficients using hard thresholding and then take inverse-DCT to get the denoised image [9].

Then, the thresholding operation (2) can be modeled as

$$Y(i,j) = \begin{cases} Y(i,j), & if\ |Y(i,j)| > T \\ 0, & if\ |Y(i,j)| \leq T \end{cases} \tag{2}$$

where T is the threshold. For this implementation, a threshold value of $T = 3.5 * \sigma$ is selected. The original signal and the hard thresholding operation are portrayed in Fig. 1.

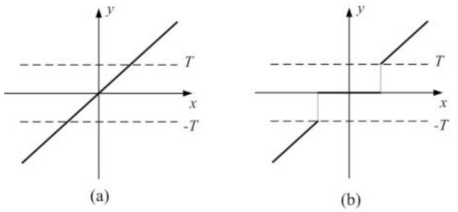

Fig. 1. Thresholding process, (a) original signal, (b) hard thresholding.

Since the present denoising algorithm uses a forward and inverse Fourier-related transform, specifically a DCT, this process becomes the primary computational module of the whole system. Thus, it has to be optimized to achieve an implementation as fast as possible.

There are several types of DCT. The most powerful and popular is two-dimensional symmetric variation of the transform that operates on 8x8 blocks (DCT8x8). The DCT8x8 is utilized in JPEG compression routines and has become a de-facto standard in image and video coding algorithms [10]. Because of GPU is a high segmented computational device, this block-division DCT approach is exploited in the present

work. Therefore, 8x8 blocks are processed in a secuential order by a single core in the serial denoising algorithm while the multithreaded and gpu versions makes a specific inter-block parallelization.

2.2 Power monitoring approach

On the power consumption measurement, we can discriminate two direct ways.

Firstly, a hardware-oriented solution, where an additional power capable device is attached between the computational equipment and the power supply. In particular, this method consists on sampling electrical current while voltage remains constant. However, voltage could be eventually measured by clamp probes. Then, power is computed by multiplying the two signals, and total energy is calculated as the integral of the power over the execution time. This approach might lead to several errors caused by clamp sampling variations and instant power aproximation (by multiplication means).

This hardware technique can be invasive or not. In the invasive way, we measure the power consumption interfering the power supply lines generally through direct current sensors (Hall effect). In contrast, an extern system measurement can be carried out by intercepting the power supply input through alternating current and voltage tools.

On the other hand, energy consumption could be supervised across particular software interfaces. In this case, real time power information could be accessed through built in on-board sensors or by a specific power estimation model. In this last approach, the power scheme is feeded by hardware counters data.

In this work, we employs a software measurement approach, where CPU power consumption is monitored through Intel RAPL interface and GPU power information is gathered using Nvidia NVML.

RAPL provides a set of counters producing energy and power consumption information. It uses a software power model that estimates energy usage by querying hardware performance counters and I/O models [11] and results are available to the user via a model specific register (MSR). This power model has been validated by Intel in [12].

NVML is an API for monitoring and managing different Nvidia GPUs features like the ability to set/unset ECC (Error Correction Code), or to monitor memory usage, temperature, utilization rates, and more. Also, this library provides the ability to query power consumption at runtime through the built in power sensor.

2.3 Measurement description

For all implementations the measuring method consists on: running the computational algorithm in one/multiple threads and the RAPL/NVML code in another thread using Pthreads, which provides negligible overhead.

Then, the only communication between the power measuring threads and computational threads is a flag variable. Furthermore, power readings are stored in a global structure. Finally, RAPL and NVML threads stop when the shared flag is reset,

which is when the CPU/GPU finalize its execution. This brief description is showed in Fig. 2.

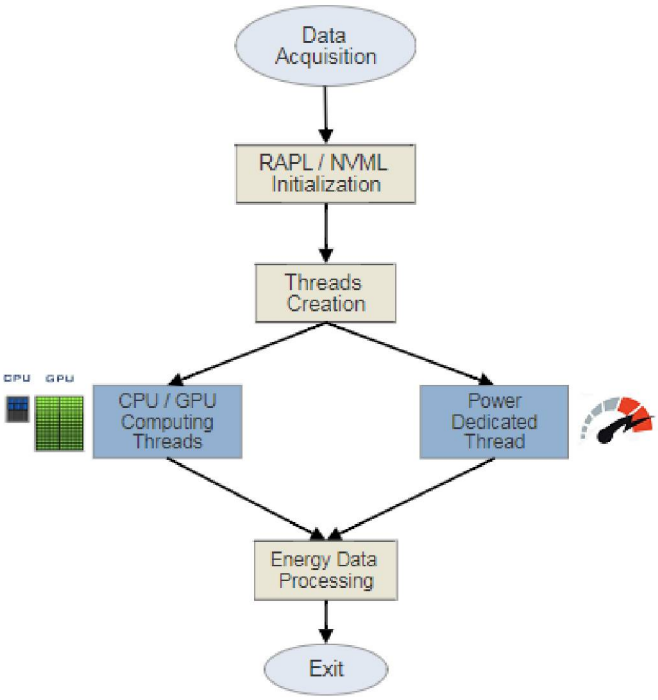

Fig. 2. Measurement method.

Besides, a critical parameter to define is the power sampling interval. Some previous research works on GPU power consumption have focused on this point. Lang and Rünger et al. [6] shows that the optimal NVML sampling frequency is 50Hz (20ms). In contrast, Burtscher et al. [13] demonstrates that the maximum frequency supported by hardware is 66.7Hz (15ms). In addition, Kasichayanula et al. [1] and Weaver et al. [14] recommends a sampling frequency of 62.5Hz (16ms).

In this work, we use a NVML (GPU) and RAPL (CPU) sampling frequency of 62.5Hz. Therefore, we wrote our own tool to query the GPU sensor via NVML interface and to obtain estimated CPU power data through RAPL.

It is also necessary to mention that NVML power information refers to whole GPU board, including DC voltage converters, integrated circuits like video chip and bridge, memories, etc. The returned value is accurate to within a range of +/- 5 milliwatts. However, Intel RAPL provides entire CPU package power data (including cores, uncore circuit and DRAM memory) with +/- 1 milliwatt precision.

Generally, power consumption of commercial CPUs and GPUs can be described as the sum of static power, dynamic power and the impact of ascending temperature. Thus,

$$P = P_S + P_D + P_T, \quad (3)$$

where P_S is static power, P_D is dynamic power and P_T is the temperature effect on power. Static power depends on chip layout and circuit technology, and is independent of workload execution. Dynamic power results from transistors switching overhead. Heat also has an impact on power due to transistors current leakage increases with temperature [15].

In this work, static power is measured when no workload is executed and while none of the CPU and GPU resources are turned off. Furthermore, temperature contribution is considered negligible because of GPU algorithm executes in a few seconds. Despite CPU code runs a longer time, the measured temperature does not increase significantly.

Finally, as dynamic power depends on the specific workload to be processed, it is the chosen variable to analyze in results section.

2.4 Target platform

The hardware platform includes a computing server with an Intel Core i3-4170 (4th gen.) processor with a 3MB cache and 2 physical cores (4 threads) at 3.70GHz. This processor is attached to a Gigabyte H81M-H motherboard, which additionally holds an 8GB RAM memory. The server also contains a Nvidia Tesla C2075 scientific computing GPU equipped with 14 multiprocessors including 32 cuda cores each (448 total cores) and a 6GB GDDR5 global memory.

Moreover, our software configuration combines a 64-bit Ubuntu distribution (Linux kernel 3.2.0) with Nvidia Driver v331.62 and CUDA Toolkit 6.0.

2.5 Experiments

As mentioned before, we implement three versions of the DCT based denoising algorithm. In particular, a serial CPU version, a multithreaded CPU version and a GPU version. The multithreaded implementation exploits the OpenMP API [16] to interpret user directives in order to create and manage threads execution.

To increase performance, both CPU versions are compiled with and without g++ optimization flags. For detailed information, go to [17].

Therefore, the five scenarios are:

- Serial version
- Serial version (compiled with optimization flags)
- Multithreaded version
- Multithreaded version (compiled with optimization flags)
- GPU version

In particular, the GPU implementation uses a 2D 8x8 threads block configuration to optimize DCT/iDCT performance. Also, we exploit shared memory and texture memory, operate with asynchronous CPU/GPU memory copies (pinned memory) and apply a multi-stream based computation.

Therefore, we took a total of 10 image datasets where all of them have a sum of 50 RGB images each. Every set is characterized by a specific image size (square or not). Such dimensions are 1MP, 2MP, 4MP, 6MP, 8MP, 14MP, 20MP, 28MP, 34MP and

70MP, where MP (megapixels) refers to the millions of pixels in the picture. In order to simulate the real acquisition, we contaminate all images with additive Gaussian white noise with an input noise level equal to 20 db.

So, we process ten image datasets through the five experiments evaluating in each set the execution time, power and energy consumption. In addition, each experiment is executed several times to obtain more accurate information.

3. Results

3.1 Denoising capability

In order to reveal the solution effectiveness, we took an RGB image from the 34MP dataset and evaluate the visual and technical output of the denoising implementation. Fig. 3 displays the obtained data.

Fig. 3. From top to bottom, left to right: original, noisy and denoised images.

In this example, the algorithm returns a highly acceptable denoised sample with a PSNR value of 33.87 dB.

3.2 Performance and Energy analysis

In this section, we introduce the evaluated computation time and some power related measurements like instant dynamic power (load dependent observed power) and total energy consumption.

As explained in measurement section, total power fluctuation associates directly with dynamic power variation. Table I shows static, dynamic and total average power for the 6th image dataset. As we mentioned, temperature impact is considered negligible in this work.

Table I. Observed power consumption (Watts)

	CPU	GPU
Static	2,61	77,98
Dynamic	8,46	51,41
Total	11,09	129,42

Static power measurements correspond with CPU and GPU idle state and it is almost constant for all dataset. Then, dynamic values vary depending on the specific workload.

So as to calculate and display dynamic power data, we use the measured static power values shown above: 2,61W for the Intel i3 CPU and 77,98W for the Nvidia Tesla GPU. These numbers are the zero reference for CPU and GPU dynamic power plot. Fig. 4 presents the execution time and instant dynamic power of the 6th image dataset for all experiments.

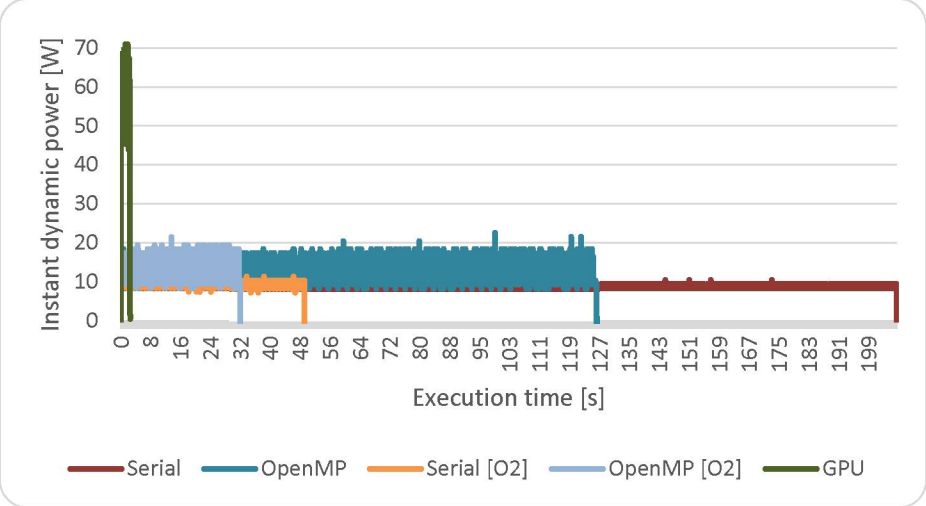

Fig. 4. Instant dynamic power.

As shown in the graph, the zero reference (idle state) represents beginning and end of algorithm execution for all cases. Notice that serial versions corresponds to less power consumption while multithreaded versions raise this parameter. Moreover, the GPU implementation presents a huge difference in power consumption regarding serial and OpenMP versions.

In contrast, GPU execution time is minimal compared to CPU versions. Fig. 5 introduces this metric for all image datasets.

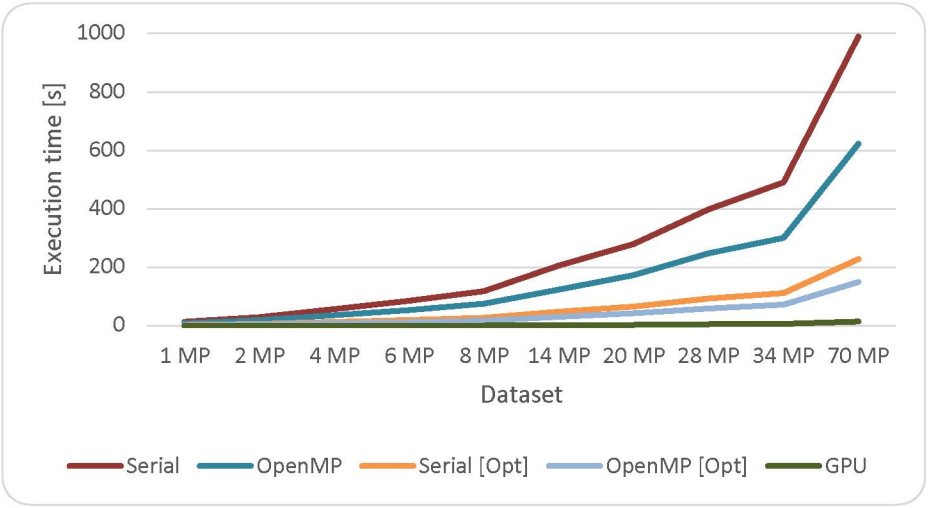

Fig. 5. Execution time.

As shown, CPU and GPU execution time increases with image size. In the first dataset, GPU outperforms multithreaded optimized CPU version by 10.2x and single-threaded optimized CPU version by 16.3x. Moreover, if these implementations are not built with compiler optimizations, the achieved speedup raises to 42.1x and 65.9x, respectively. These speedup values remain almost constant throughout all image datasets.

Having exposed instant power measurements and total execution time, we can now present the average energy consumption for all solutions. This metric can be computed by multiplying average power consumption (static + dynamic) over total execution time.

Whereas measured power consumption for all experiments remains almost constant, it is probably that energy data will present a nearly linear behavior over the performance one. Then, energy consumption and execution time curves will be proportional and the offset between them will be set by power consumption values.

Fig. 6 shows the average energy consumption for all experiments.

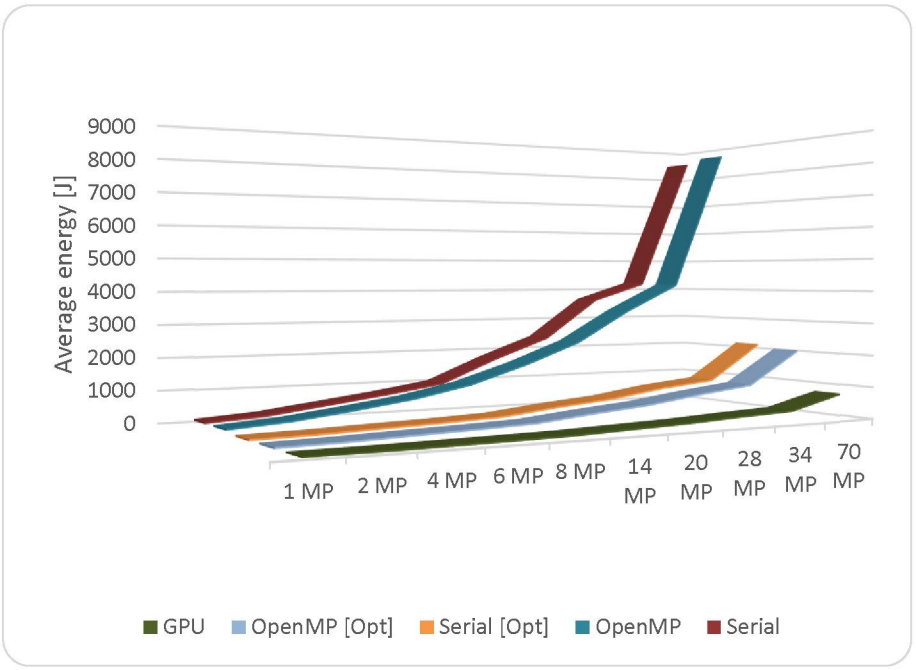

Fig. 6. Energy consumption.

GPU energy consumption is significantly lower that CPU solutions. For instance, in the 70MP dataset the multithreaded optimized CPU implementation returns an average energy value equal to 2068J while the GPU average energy consumption drops to 809J (2.5x). Further, GPU outperforms the single-threaded optimized version by 2.7x. Similarly, when using no compiler optimizations, the GPU beat CPU applications by 10.6x and 10.4x in total energy consumption.

4. Conclusion and future work

This article presents some measurements and a detailed analysis of the performance and energy consumption of a DCT based denoising algorithm. We implement three different codes including a single-threaded CPU version, a multithreaded CPU version and a GPU version. CPU implementations also include a compiler-optimized variant.

We showed that GPU accelerated code can outperform CPU serial and multithreaded programs in terms of performance and total energy consumption. Also, we expose that compiler based optimizations are a vital resource for lower CPU execution time and energy consumption. However, this optimized GPU implementation can offset CPU execution time by almost 66x. Despite GPU measured power is huge, applications can finish faster so the total energy consumption is notably less than CPU versions.

In future work, we plan to combine software based power data with physical measurements directly from CPU and GPU. Therefore, this accurate information combined with performance counters data can be the starting point to design a new model to predict CPU and GPU power consumption.

5. References

[1] K. Kasichayanula, D. Terpstra. Power Aware Computing on GPUs. In: Symposium on Application Accelerators in High Performance Computing (2012).

[2] X. Ma, M. Dong, L. Zhong, Z. Deng. Statistical Power Consumption Analysis and Modeling for GPU-based Computing. In: HotPower ACM SOSP Workshop Power Aware Computing and Systems (2009).

[3] H. Nagasaka, N. Maruyama. Statistical Power Modeling of GPU Kernels Using Performance Counters. In: International Conference on Green Computing (2010).

[4] R. Suda, Da Qi Ren. Accurate Measurements and Precise Modeling of Power Dissipation of CUDA Kernels toward Power Optimized High Performance CPU-GPU Computing. In: International Conference on Parallel and Distributed Computing, Applications and Technologies (2009).

[5] S. Huang, S. Xiao, W. Feng. On the energy efficiency of Graphic Processing Units for scientific computing. In: IEEE Symposium on Parallel & Distributed Processing (2009).

[6] J. Lang, G. Rünger. High-Resolution Power Profiling of GPU Functions Using Low-Resolution Measurement. In: Euro-Par 2013 Parallel Processing Lecture Notes in Computer Science Volume 8097, Springer, pp 801–812 (2013).

[7] NVML Reference Guide, http://docs.nvidia.com/deploy/nvml-api/index.html.

[8] Signal denoising, http://eeweb.poly.edu/iselesni/DoubleSoftware/signal.html.

[9] M. Biswas, H. Om. A New Soft-Thresholding Image Denoising Method. In: 2nd International Conference on Communication, Computing & Security (2012).

[10] A. Obukhov, A. Kharlamov. Discrete Cosine Transform for 8x8 blocks with CUDA. Nvidia whitepaper (2008).

[11] Running Average Power Limit – RAPL, https://01.org/blogs/2014/running-average-power-limit-%E2%80%93-rapl.

[12] E. Rotem, A. Naveh, D. Rajwan, A. Anathakrishnan, E. Weissmann, Power-management architecture of the Intel microarchitecture codenamed Sandy Bridge. In: IEEE Micro, vol. 32, no. 2, pp. 20–27, 2012.

[13] M. Burtscher, I. Zecena, Z. Zong. Measuring GPU Power with the K20 Built-in Sensor. In: GPGPU-7 Proceedings of Workshop on General Purpose Processing Using GPUs (2014).

[14] V. Weaver, M. Johnson, K. Kasichayanula, J. Ralph, P. Luszczek, D. Terpstra, S. Moore. Measuring Energy and Power with PAPI. In: 41st International Conference on Parallel Processing Workshops (2012).

[15] D. Li, S. Byna, S. Chakradhar. Energy-Aware Workload Consolidation on GPU.

[16] OpenMP Application Programming Interface, http://www.openmp.org/wp-content/uploads/openmp-4.5.pdf.

[17] GCC Command Options: Options That Control Optimization, https://gcc.gnu.org/onlinedocs/gcc-4.6.2/gcc/Optimize-Options.html.